

EFFICIENT RANDOMIZED ALGORITHMS FOR ADAPTIVE LOW-RANK FACTORIZATIONS OF LARGE MATRICES

YU GU*, WENJIAN YU†, AND YAOHANG LI‡

Abstract. In this paper, randomized techniques for computing low-rank factorizations are presented. The proposed methods take in a tolerance ε and an $m \times n$ matrix \mathbf{A} , and output an approximate low-rank factorization of \mathbf{A} , whose error measured in the Frobenius norm is within ε . The techniques are based on the blocked randQB scheme proposed by P.-G. Martinsson and S. Voronin, producing a QB factorization. By employing an economic error indicator and moving \mathbf{A} out of the loop, the techniques result in two algorithms called randQB.EI and randQB.FP. They are mathematically equivalent to the existing blocked scheme, but are more computationally efficient. The randQB.FP algorithm also owns the merit of pass-efficiency. Numerical experiments on a multi-core parallel computing server show that the proposed algorithms have the similar accuracy as the blocked randQB scheme, but cost a small fraction of runtime and memory. The benefits are even larger (up to 20X) for handling large sparse matrices. Compared with the adaptive range finder algorithm, the proposed methods output much smaller and close to optimal rank while satisfying the preset accuracy tolerance.

Key words. adaptive rank determination, low-rank factorization, parallel algorithm, single-pass algorithm, randomized algorithm, standard Gaussian matrix

AMS subject classifications. 15A18, 65F30, 65F15, 68W20, 60B20

1. Introduction. Low-rank matrix factorizations, like the partial singular value decomposition (SVD) and the rank-revealing QR factorization, play a crucial role in data analysis and scientific computing. In recent years, techniques based on randomization has been investigated for performing the computation and low-rank factorization of large matrices [1, 2, 3, 8, 9, 11, 17, 18]. They exploit modern computing architectures more efficiently than classical methods, and exhibit large potential for dealing with truly massive data sets.

The basic idea of the randomized techniques is using random sampling to identify the subspace capturing the dominant actions of a matrix. For an $m \times n$ matrix \mathbf{A} , suppose this k -dimensional dominant subspace has a set of orthogonal basis vectors forming an orthonormal matrix \mathbf{Q} (here $k < \min(m, n)$). Because the columns of \mathbf{Q} approximate the major basis vectors of $\text{range}(\mathbf{A})$, we have [1, 2]:

$$(1) \quad \mathbf{A} \approx \mathbf{Q}\mathbf{B},$$

where \mathbf{B} is a $k \times n$ matrix, and

$$(2) \quad \mathbf{B} = \mathbf{Q}^T \mathbf{A}.$$

Then, standard factorizations can be performed on the smaller matrix \mathbf{B} , and their results are used to obtain the low-rank factorizations of \mathbf{A} . The randomized algorithm involves the same or fewer floating-point operations (*flops*) than classical algorithms, and is more robust and more efficient by exploiting modern computing architectures.

*Department of Computer Science and Technology, Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100084, China (guyul3@mails.tsinghua.edu.cn).

†Corresponding author. Department of Computer Science and Technology, Tsinghua National Lab of Information Science and Technology, Tsinghua University, Beijing 100084, China (yu-wj@tsinghua.edu.cn).

‡Department of Computer Science, Old Dominion University, Norfolk, VA 23529, USA (yaochang@cs.odu.edu).

The basic randQB procedure
Input: \mathbf{A} , k , s
Output: \mathbf{Q} , \mathbf{B} .
(1) $\mathbf{\Omega} = \text{randn}(n, k + s)$
(2) $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{\Omega})$
(3) $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$

FIG. 1. The basic randomized procedure for producing a rank- k approximation [2]. The over-sampling strategy is employed with a small integer s .

It also tends to be a kind of *pass-efficient* algorithm, which requires only a constant number of passes over the data, as opposed to $O(k)$ passes for classical algorithms. This is very desirable for the situation where the data are huge in size. The single-pass algorithms are particularly useful for data generated in a streaming fashion or cannot be stored in fast memory [2, 10].

The approximation presented by (1) and (2) can be regarded as a kind of low-rank factorization of \mathbf{A} . We call it QB factorization or QB approximation, which is the object of this work. In [1, 2], a basic randomized scheme for computing the QB approximation was presented, as that shown in Figure 1. For producing close to the optimal rank- k SVD, the over-sampling scheme using more than k columns in the random Gaussian matrix $\mathbf{\Omega}$ is actually employed [2]. Here, s stands for the over-sampling parameter. The produced \mathbf{Q} matrix has $l = k + s$ columns. “ $\text{orth}(\mathbf{X})$ ” denotes orthonormalization of the columns of \mathbf{X} . In practice, it is achieved efficiently by a call to a packaged QR factorization (eg. the `qr` command in Matlab), which implements the QR factorization *without pivoting*. We call this basic procedure “randQB”.

The basic randQB procedure could not produce the optimal low-rank approximation as the approach based on the SVD of \mathbf{A} . However, in many applications the optimal approximation is not necessary, and even impossible to obtain due to the huge computational cost of performing SVD for a large matrix. The existing works have revealed that this randomized algorithm could produce a good enough solution.

The existing randomized factorization techniques aim at two problems:

- The *fixed-rank* problem, where the rank of the output factor matrices is given.
- The *auto-rank* problem, where the rank of factor matrices needs to be automatically determined by taking into account some accuracy conditions.

The procedure in Figure 1 is for the fixed-rank problem. The auto-rank problem is also called *fixed-precision* problem [2]. Taking the QB factorization as an example, we shall seek \mathbf{Q} and \mathbf{B} with suitable rank such that

$$(3) \quad \|\mathbf{A} - \mathbf{QB}\| < \varepsilon,$$

where ε is a given accuracy tolerance. More practically, the relative accuracy tolerance should be considered, where ε is a fraction of $\|\mathbf{A}\|$, i.e. $\delta\|\mathbf{A}\|$.

For the fixed-rank problem, the randomized algorithm exhibits advantages over the classical algorithms. Compared with the rank-revealing QR factorization [4, 5] for low-rank approximation, it has lower computational cost and can obtain substantial speedup on a parallel computing platform [18].

For the auto-rank problem, an adaptive randomized range finder algorithm was proposed in [2]. It employs the incremental sampling approach with a probabilistic error estimator to determine the rank satisfying the accuracy condition. However, the theoretic error bound for building this error estimator is loose, causing the rank largely overestimated. Besides, it measures the error in 2-norm, instead of the widely used Frobenius norm. Therefore, it is neither efficient nor convenient for many practical applications. Recently, Martinsson and Voronin proposed a randomized blocked algorithm for computing rank-revealing factorizations [1]. It incrementally produces

the QB factorization based on the combination of column pivoting Gram-Schmidt scheme [4, 5], randomized sampling, and the *blocking* for high performance. And, it has a different stopping criterion including an explicit approximation error, instead of the probabilistic error estimator. This makes it natural to incorporate adaptive rank determination for the auto-rank problem. However, the algorithm in [1] is more likely for the rank-revealing factorization of a small- or medium-size matrix, instead of the low-rank approximation of a large matrix. This is due to the fact that maintaining the explicit error matrix is costly in runtime and memory usage, and also not feasible for the large and possibly sparse matrices in practical scenarios.

The aim of this work is to provide a solution to the auto-rank problem of low-rank QB factorization, and we focus on using the practical Frobenius norm to measure the approximation error. Improvements are made on the algorithm of [1] for pursuing the adaptability for large, sparse matrices, and higher computational efficiency. Firstly, an economic error indicator is presented which replaces the explicitly generated error matrix. This saves the computational cost and memory usage. Secondly, we reorganize the algorithm to move the matrix multiplications out of the loop so as to obtain more performance benefit from high-level BLAS operation. It also owns the merit of only involving a single pass over matrix \mathbf{A} . Based on them, two algorithms with names `randQB_EI` and `randQB_FP` are derived, which are proved to be mathematically equivalent to the blocked `randQB` algorithm in [1]. To enhance the accuracy, the power schemes for the both algorithms are also developed. They inherit the algorithms' merits, and make efficient treatment for large, sparse matrices in actual applications. Compared with the adaptive randomized range finder in [2], the proposed algorithms utilize a precise deterministic stopping criterion, and therefore avoid rank overestimation. Numerical experiments on a multi-core computer are carried out to demonstrate the efficiency and accuracy of the proposed algorithm, and its effectiveness for adaptive rank determination.

2. Technical preliminaries. This section summarizes the background we need for presenting the proposed algorithms in sections 3-5. We briefly introduce standard matrix factorizations in section 2.1, and review the related randomized algorithms in section 2.2. Lastly, the orthogonal projector matrix and its properties are introduced, providing theoretic support for the proposed algorithms.

Throughout the paper, we measure vectors with their Euclidean norm (2-norm). Two kinds of matrix norm are usually considered: Frobenius norm and spectral norm (2-norm). Basically, the spectral norm of a matrix is difficult to calculate because it involves the computation of the matrix's largest singular value. The Frobenius norm $\|\mathbf{A}\| = (\sum_{i,j} |\mathbf{A}(i,j)|^2)^{1/2}$ is much easier for calculation, and is therefore more widely used in various applications. We measure matrices with their Frobenius norm by default. We also assume that all matrices are real, although the generalization to complex matrices is of no difficulty.

2.1. Matrix factorizations. Computing the low-rank matrix factorizations or the rank-revealing factorizations can be realized using the standard techniques like partial singular value decomposition (SVD) or partial QR factorization.

Let \mathbf{A} denote an $m \times n$ matrix. Then, the (economic) SVD is

$$(4) \quad \mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

where \mathbf{U} is an $m \times \min(m, n)$ orthonormal matrix, \mathbf{V} is an $n \times \min(m, n)$ orthonormal matrix, and $\mathbf{\Sigma}$ is a nonnegative diagonal matrix. The diagonal entries of $\mathbf{\Sigma}$ are the

singular values of \mathbf{A} . The columns of matrices \mathbf{U} and \mathbf{V} are the left and right singular vectors of \mathbf{A} , respectively. Usually, the columns of \mathbf{U} and \mathbf{V} are arranged so that the singular values are sorted in a descending order along the diagonal of $\mathbf{\Sigma}$. This means $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(m,n)} \geq 0$, if σ_j denotes the diagonal entry $\mathbf{\Sigma}(j, j)$.

Taking the first k , $k < \min(m, n)$, columns of \mathbf{U} and \mathbf{V} respectively, and the first k singular values in $\mathbf{\Sigma}$, we have the partial SVD of matrix \mathbf{A} :

$$(5) \quad \mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T \approx \mathbf{A},$$

where \mathbf{U}_k and \mathbf{V}_k are the orthonormal matrices, including the first k left and right singular vectors, respectively. $\mathbf{\Sigma}_k$ is the diagonal matrix identical to the $k \times k$ upper-left-corner submatrix of $\mathbf{\Sigma}$. The result of partial SVD, \mathbf{A}_k , is actually the optimal rank- k approximation of \mathbf{A} . The Eckart-Young theorem [6] states that in the spectral norm and Frobenius norm, the error of \mathbf{A}_k to \mathbf{A} is always minimal,

$$(6) \quad \|\mathbf{A}_k - \mathbf{A}\| = \min_{\text{rank}(\tilde{\mathbf{A}})=k} \|\tilde{\mathbf{A}} - \mathbf{A}\|.$$

It can be further proved that [7]:

$$(7) \quad \|\mathbf{A}_k - \mathbf{A}\|_2 = \sigma_{k+1}, \text{ and } \|\mathbf{A}_k - \mathbf{A}\| = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2}.$$

Any $m \times n$ matrix \mathbf{A} admits an (economic) QR factorization:

$$(8) \quad \mathbf{A}\mathbf{P} = \mathbf{Q}\mathbf{R},$$

where \mathbf{Q} is an $m \times \min(m, n)$ orthonormal matrix, \mathbf{R} is a $\min(m, n) \times n$ upper triangular matrix, and \mathbf{P} is an $n \times n$ permutation matrix. The QR factorization is often computed through column pivoting combined with the Householder transformation, Gram-Schmidt process, or Givens rotation transformation [5]. \mathbf{P} records the result of column exchanges made by the pivoting operations. The QR factorization is performed in an incremental manner, and can be stopped after the first k columns ($k < \min(m, n)$) of \mathbf{R} have been computed. This obtains a partial QR factorization:

$$(9) \quad \mathbf{A}\mathbf{P}_k \approx \mathbf{Q}_k \mathbf{R}_k,$$

where \mathbf{Q}_k is an $m \times k$ orthonormal matrix, \mathbf{R}_k is a $k \times n$ upper triangular matrix, and \mathbf{P}_k is an $n \times n$ permutation matrix. The first k columns of \mathbf{R}_k is just the $k \times k$ upper-left-corner submatrix of \mathbf{R} in the full QR factorization (8).

Both (5) and (9) give a rank- k approximation of matrix \mathbf{A} . By choosing a suitable truncation or stopping criterion, both factorizations can be utilized to reveal the rank of \mathbf{A} . To make k a good estimation of the rank, we shall ensure $\sigma_{k+1} \approx 0$ for the partial SVD, and the rows below the k -th row in \mathbf{R} are almost zeros for the partial QR factorization. For low-rank approximation, where k may be much smaller than the rank of \mathbf{A} , the result of partial QR factorization is not optimal.

From the viewpoint of computation, the partial QR factorization costs typically $O(kmn)$ flops for a dense matrix \mathbf{A} [4]. In contrast, computing the partial SVD for a dense \mathbf{A} is more expensive, because it usually relies on constructing the full SVD with $O(mnp)$ flops, where $p = \min(m, n)$. For a sparse \mathbf{A} , the Krylov subspace method may be economic for computing the partial SVD. But this is limited to the situation

where only a very small number of singular values are of interest. Otherwise, the Krylov subspace method is expensive due to its slow convergence.

It should be mentioned that the partial QR factorization can be computed in an incremental manner (through Gram-Schmidt procedure), which makes it suitable for adaptive rank determination. Its drawback is due to the difficulty to obtain substantial speedup on a parallel computing platform, as mentioned in [1].

2.2. Randomized algorithms. To make a rank- k factorization for an $m \times n$ matrix \mathbf{A} , with the basic randQB procedure in Figure 1 we obtain an $m \times l$ orthonormal matrix \mathbf{Q} , where $l > k$ due to the over-sampling. Then, the $l \times n$ matrix \mathbf{B} is computed by $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$, for minimizing the approximation error. With this QB approximation, the standard factorizations can be efficiently computed. For example, we perform the SVD on the $l \times n$ matrix \mathbf{B} to obtain a factorization $\mathbf{B} = \tilde{\mathbf{U}} \tilde{\mathbf{\Sigma}} \tilde{\mathbf{V}}^T$. Then,

$$(10) \quad \mathbf{A} \approx \mathbf{Q}\mathbf{B} = \mathbf{Q}\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T.$$

Now, we can choose the first k columns of matrices $\mathbf{Q}\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ respectively, and the $k \times k$ upper-left corner submatrix of $\tilde{\mathbf{\Sigma}}$. They approximate the rank- k SVD factors in (5). Similarly, by changing the factorizations made on \mathbf{B} we can obtain the approximate QR factorization and CUR factorization, etc [1].

The output of the randomized approximation algorithms is a random variable, as it depends on the drawing of a Gaussian matrix. It has been proven that the variation in this random variable is small, which means the output is always very close to the variable's expectation. For more details, please refer to [1, 2]. The bound of the expectation of the approximation error has been derived in [2]. And, the distribution of errors caused by randomness has been demonstrated in [1].

For the auto-rank problem, the adaptive randomized range finder [2] employs a posteriori error estimation. It is based on the statement that

$$(11) \quad \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T \mathbf{A}\|_2 \leq 10 \sqrt{\frac{2}{\pi}} \max_{i=1, \dots, r} \|(\mathbf{A} - \mathbf{Q}\mathbf{Q}^T \mathbf{A})\boldsymbol{\omega}^{(i)}\|,$$

with probability at least $1 - 10^{-r}$. Here $\|\cdot\|_2$ stands for the spectral norm, $\boldsymbol{\omega}^{(i)}$ is a random vector, and r is a small integer, e.g. $r = 10$.

The technique in [1] is based on a greedy Gram-Schmidt procedure. It is shown in Figure 2(a), which constitutes the “single-vector randQB” algorithm for computing the QB approximation. If lines (4) and (5) are replaced such that we pick \mathbf{q}_i as the largest column of $\mathbf{A}^{(i)}$, the algorithm becomes the column pivoted Gram-Schmidt algorithm for partial QR factorization. It is straight-forward to show that if the algorithm is executed in exact arithmetic, \mathbf{Q}_i is orthonormal, $\mathbf{B}_i = \mathbf{Q}_i^T \mathbf{A}$, and

$$(12) \quad \mathbf{A}^{(i)} = \mathbf{A} - \mathbf{Q}_i \mathbf{B}_i.$$

Therefore, the error of QB approximation is updated, allowing a precise stopping criterion for the auto-rank problem. The algorithm is mathematically equivalent to the basic randQB procedure shown in Figure 1, except for the stopping criterion used. In order to exploit *blocking* to attain high performance of linear algebraic computation, the single-vector randQB algorithm is converted to a blocked version, i.e. the blocked randQB algorithm in Figure 2(b). Note that matrix \mathbf{A} is overwritten in line (6), and the re-orthogonalization operation in line (4) is for easing the accumulation of round-off error under floating point arithmetic. The experiments showed that the

The single-vector randQB algorithm	The blocked randQB algorithm
Input: \mathbf{A}, ε Output: $\mathbf{Q}_k, \mathbf{B}_k$. (1) $\mathbf{Q}_0 = [\]; \mathbf{B}_0 = [\]; \mathbf{A}^{(0)} = \mathbf{A}; i = 0;$ (2) while $\ \mathbf{A}^{(i)}\ > \varepsilon$ do (3) $i = i + 1$ (4) $\boldsymbol{\omega}_i = \text{randn}(n, 1)$ (5) $\mathbf{q}_i = \mathbf{A}^{(i-1)}\boldsymbol{\omega}_i; \mathbf{q}_i = \mathbf{q}_i / \ \mathbf{q}_i\ ;$ (6) $\mathbf{b}_i = \mathbf{q}_i^T \mathbf{A}^{(i-1)}$ (7) $\mathbf{Q}_i = [\mathbf{Q}_{i-1}, \mathbf{q}_i]$ (8) $\mathbf{B}_i = \begin{bmatrix} \mathbf{B}_{i-1} \\ \mathbf{b}_i \end{bmatrix}$ (9) $\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)} - \mathbf{q}_i \mathbf{b}_i$ (10) end while	Input: $\mathbf{A}, \varepsilon, b$ Output: \mathbf{Q}, \mathbf{B} . (1) for $i = 1, 2, 3, \dots$ (2) $\boldsymbol{\Omega}_i = \text{randn}(n, b)$ (3) $\mathbf{Q}_i = \text{orth}(\mathbf{A}\boldsymbol{\Omega}_i)$ (4) $\mathbf{Q}_i = \text{orth}(\mathbf{Q}_i - \sum_{j=1}^{i-1} \mathbf{Q}_j \mathbf{Q}_j^T \mathbf{Q}_i)$ (5) $\mathbf{B}_i = \mathbf{Q}_i^T \mathbf{A}$ (6) $\mathbf{A} = \mathbf{A} - \mathbf{Q}_i \mathbf{B}_i$ (7) if $\ \mathbf{A}\ < \varepsilon$ then stop (8) end for (9) $\mathbf{Q} = [\mathbf{Q}_1 \dots \mathbf{Q}_i]; \mathbf{B} = [\mathbf{B}_1^T \dots \mathbf{B}_i^T]^T.$
(a) The randQB_sv algorithm	(b) The randQB_b algorithm

FIG. 2. The randomized algorithms for QB factorization proposed in [1].

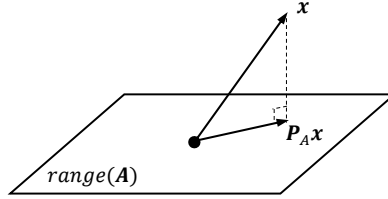


FIG. 3. The geometric explanation of the orthogonal projector matrix \mathbf{P}_A .

blocked randQB algorithm has the same or better accuracy than the column-pivoted QR factorization, and runs much faster on multi-core architectures.

Due to the precise error calculation, the blocked randQB algorithm is more suitable for the auto-rank problem than the adaptive randomized ranger finder [2]. However, the explicit expression of the error matrix, i.e. line (6) in Figure 2(b), causes difficulties when applying the algorithm to a large matrix \mathbf{A} . If \mathbf{A} is a sparse matrix, the sparsity will be destroyed, causing large runtime and memory cost.

2.3. Orthogonal projectors. The concept of orthogonal projector matrix is used in this paper as well as many preceding works. An *orthogonal projector* is a matrix representing the linear transformation which converts any vector to its orthogonal projection on a subspace. It is uniquely determined by the subspace. Figure 3 shows the orthogonal projection transformation to $\text{range}(\mathbf{A})$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$. The corresponding orthogonal projector is denoted by \mathbf{P}_A . Based on the theory of linear least squares, if \mathbf{A} has linearly independent columns,

$$(13) \quad \mathbf{P}_A = \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T.$$

If \mathbf{A} is an orthonormal matrix, the expression is simplified to:

$$(14) \quad \mathbf{P}_A = \mathbf{A} \mathbf{A}^T.$$

Figure 3 illustrates the following property of \mathbf{P}_A :

$$(15) \quad \forall \mathbf{x} \in \mathbb{R}^m, \mathbf{P}_A \mathbf{x} \in \text{range}(\mathbf{A}), \quad \text{and if } \mathbf{x} \in \text{range}(\mathbf{A}), \mathbf{P}_A \mathbf{x} = \mathbf{x}.$$

Obviously, $\text{range}(\mathbf{P}_A) = \text{range}(\mathbf{A})$. From (13), it is easy to verify that the orthogonal projector is a symmetric matrix, and $\mathbf{P}_A^2 = \mathbf{P}_A$. Based on its geometric meaning, it is easy to see that the orthogonal projector determined by the orthogonal complement of $\text{range}(\mathbf{A})$ is $\mathbf{I} - \mathbf{P}_A$, where \mathbf{I} is the identity matrix.

Below are some results about the orthogonal projector matrix.

LEMMA 1. For $\mathbf{A} \in \mathbb{R}^{m \times n}$, if \mathbf{A} has linearly independent columns,

$$(16) \quad \mathbf{P}_A \mathbf{A} - \mathbf{A} = \mathbf{O},$$

where \mathbf{O} stands for the zero matrix.

Proof. Plugging in $\mathbf{P}_A = \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$, the result follows immediately. \square

LEMMA 2. For $\mathbf{A} \in \mathbb{R}^{m \times n}$, orthonormal matrix $\mathbf{Q} \in \mathbb{R}^{m \times k}$ and $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$,

$$(17) \quad (\mathbf{A} - \mathbf{QB})^T (\mathbf{A} - \mathbf{QB}) = \mathbf{A}^T \mathbf{A} - \mathbf{B}^T \mathbf{B}.$$

Proof.

$$\begin{aligned} (\mathbf{A} - \mathbf{QB})^T (\mathbf{A} - \mathbf{QB}) &= (\mathbf{A} - \mathbf{QQ}^T \mathbf{A})^T (\mathbf{A} - \mathbf{QQ}^T \mathbf{A}) \\ &\stackrel{1}{=} (\mathbf{A} - \mathbf{P}_Q \mathbf{A})^T (\mathbf{A} - \mathbf{P}_Q \mathbf{A}) \\ &= \mathbf{A}^T (\mathbf{I} - \mathbf{P}_Q^T) (\mathbf{I} - \mathbf{P}_Q) \mathbf{A} \\ &\stackrel{2}{=} \mathbf{A}^T (\mathbf{I} - \mathbf{P}_Q)^2 \mathbf{A} \\ &\stackrel{3}{=} \mathbf{A}^T (\mathbf{I} - \mathbf{P}_Q) \mathbf{A} \\ &= \mathbf{A}^T (\mathbf{I} - \mathbf{QQ}^T) \mathbf{A} \\ &= \mathbf{A}^T \mathbf{A} - \mathbf{B}^T \mathbf{B}. \end{aligned}$$

Here, \mathbf{P}_Q stands for the orthogonal projector determined by $\text{range}(\mathbf{Q})$. Equality 1 comes from (14). Note that $\mathbf{I} - \mathbf{P}_Q$ is also an orthogonal projector (to an orthogonal complement subspace). Equalities 2 and 3 are by the properties of orthogonal projector \mathbf{P} : $\mathbf{P}^T = \mathbf{P}$ and $\mathbf{P}^2 = \mathbf{P}$, respectively. \square

LEMMA 3. For $\mathbf{A} \in \mathbb{R}^{m \times n}$, orthonormal matrix $\mathbf{Q} \in \mathbb{R}^{m \times k}$ and $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$,

$$(18) \quad \|\mathbf{A} - \mathbf{QB}\|^2 = \|\mathbf{A}\|^2 - \|\mathbf{B}\|^2.$$

Proof. Due to the property of Frobenius norm of a matrix, we know for a matrix \mathbf{M} , $\|\mathbf{M}\|^2 = \text{tr}(\mathbf{M}^T \mathbf{M})$. Here, $\text{tr}(\cdot)$ calculates the trace of a matrix. Then, we apply the trace operation to both sides of (17), we can prove (18). \square

In both `randQB_sv` and `randQB_b` algorithms, matrices \mathbf{Q} and \mathbf{B} are built incrementally. This means $\|\mathbf{B}\|$ can be evaluated incrementally, and further the approximation error $\|\mathbf{A} - \mathbf{QB}\|$ can be calculated incrementally based on Lemma 3.

3. An algorithm without the explicit error matrix. Algorithm 1 is derived from the `randQB_b` algorithm, where the approximation error matrix is removed. It is easy to find out that lines 4 ~ 7 of Algorithm 1 is equivalent to lines (2) ~ (5) of the `randQB_b` algorithm in Figure 2(b). This means Algorithm 1 also produces the QB factorization. To strictly prove the equivalence of the stopping criteria of the both algorithms, we first clarify the notations. In this section and all those follow, we use $v^{(i)}$ to denote the value of any variable v after the i -th iteration of the loop is executed. Moreover, we assume all sample matrix $\mathbf{\Omega}_i$ is of full column rank.

PROPOSITION 1. $\forall i$, $E^{(i)}$ in Algorithm 1 satisfies

$$(19) \quad E^{(i)} = \|\mathbf{A} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)}\|^2.$$

Proof. Based on line 10 and the property of Frobenius norm,

$$(20) \quad E^{(i)} = \|\mathbf{A}\|^2 - \sum_{j=1}^i \|\mathbf{B}_j\|^2 = \|\mathbf{A}\|^2 - \|\mathbf{B}^{(i)}\|^2.$$

Then, with Lemma 3, we can see that (19) holds. \square

Algorithm 1 Auto-rank QB factorization with the explicit error matrix removed

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, desired accuracy tolerance ε , block size b .

Output: \mathbf{Q} , \mathbf{B} , s.t. $\|\mathbf{A} - \mathbf{QB}\| < \varepsilon$.

```

1:  $\mathbf{Q} = []$ ;  $\mathbf{B} = []$ ;
2:  $E = \|\mathbf{A}\|^2$ 
3: for  $i = 1, 2, 3, \dots$  do
4:    $\mathbf{\Omega}_i = \text{randn}(n, b)$ 
5:    $\mathbf{Q}_i = \text{orth}(\mathbf{A}\mathbf{\Omega}_i - \mathbf{Q}(\mathbf{B}\mathbf{\Omega}_i))$ 
6:    $\mathbf{Q}_i = \text{orth}(\mathbf{Q}_i - \mathbf{Q}(\mathbf{Q}^T \mathbf{Q}_i))$ 
7:    $\mathbf{B}_i = \mathbf{Q}_i^T \mathbf{A} - \mathbf{Q}_i^T \mathbf{QB}$ 
8:    $\mathbf{Q} = [\mathbf{Q}, \mathbf{Q}_i]$ 
9:    $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_i \end{bmatrix}$ 
10:   $E = E - \|\mathbf{B}_i\|^2$ 
11:  If  $\sqrt{E} < \varepsilon$  then stop
12: end for
```

From this proposition, we see that E is actually the error indicator identical to square of the approximation error. It is updated in the iteration by calculating the Frobenius norm of \mathbf{B}_i . So, it avoids generating the dense error matrix in `randQB_b`, and can be computed more efficiently in case that matrix \mathbf{A} is large or sparse. This `randQB` algorithm with an economic error indicator is denoted by `randQB_EI`.

Besides, lines 10 and 11 in Algorithm 1 can be replaced by a row-by-row scheme. Once the accuracy tolerance is attained with certain rows of \mathbf{B}_i , we can stop the computation. It makes the output rank an arbitrary integer, instead of b 's multiple.

It should be pointed out that the incremental error indicator only works for the matrix Frobenius norm. If the matrix has to be measured in spectral norm, we can use the fact that the Frobenius norm is an upper bound of the spectral norm and perform post-processing steps after Algorithm 1, as suggested in [1].

3.1. Analysis of computational cost. Same as [1], we can compare the computational cost for different randomized QB-factorization algorithms. We assume multiplying two (dense) matrices of sizes $m \times n$ and $n \times r$ costs $C_{mm}mnr$ flops, while performing a QR factorization of a matrix of size $m \times n$ costs $C_{qr}mn \min(m, n)$ flops. With these two scaling constants: C_{mm} and C_{qr} , we first roughly estimate the computational cost for the situation where \mathbf{A} is a *dense* matrix.

Using T_{randQB} and T_{randQB_b} to denote the runtime for the algorithms `randQB` (Figure 1) and `randQB_b` (Figure 2(b)) respectively, we rewrite the results from [1].

$$(21) \quad T_{randQB} \sim 2C_{mm}mnl + C_{qr}ml^2.$$

$$(22) \quad T_{randQB_b} \sim 3C_{mm}mnl + C_{mm}ml^2 + \frac{2}{t}C_{qr}ml^2,$$

where l is the number of columns in the resulting matrix \mathbf{Q} , and t satisfies $l = tb$. Note that l can be a little bit larger than k if the rank- k SVD is pursued.

For Algorithm 1, the runtime is roughly:

$$(23) \quad \begin{aligned} T_{randQB_EI} &\sim 2C_{mm}mnl + C_{mm}(4m + 2n)b^2 \sum_{i=1}^{t-1} i + \frac{2}{t}C_{qr}ml^2 \\ &\sim 2C_{mm}mnl + C_{mm}(2m + n)l^2 + \frac{2}{t}C_{qr}ml^2. \end{aligned}$$

Because l is usually much smaller than m and n , we see that the flop count of Algorithm 1 is about 2/3 that of blocked randQB algorithm. And, it is comparable to that of the basic randQB procedure.

In practice, a large matrix is often a sparse one. For such situation, Algorithm 1 is much better than the blocked randQB algorithm [1]. With Algorithm 1, the sparsity of \mathbf{A} can be utilized to reduce the matrix computations. In contrast, this cannot be taken advantage of by the algorithms in Figure 2, as the sparsity of $\mathbf{A}^{(i)}$ quickly loses after the first iteration. Besides, since the error matrix is not stored, the memory saving of Algorithm 1 becomes significant if \mathbf{A} is of large size.

4. An algorithm with fewer passes over matrix \mathbf{A} . In this section, we derive a single-pass algorithm for the QB factorization, by moving the multiplications with \mathbf{A} out of the loop. We first present the scheme without the re-orthogonalization step, i.e. step 6 in Algorithm 1. Then, the version with re-orthogonalization is given.

4.1. The version without re-orthogonalization. The algorithm is presented as Algorithm 2. Its key idea is to pre-compute the products of \mathbf{A} and random vectors, as well as $\mathbf{A}^T \mathbf{A}$ and the random vectors. Then, in the loop we take columns from the pre-computed $\mathbf{\Omega}$, \mathbf{G} and \mathbf{H} matrices, to get the needed information. For example, with \mathbf{G} the 8th and 9th lines in Algorithm 2 perform the same function as line (3) in the randQB.b algorithm. The equivalence between line 10 in Algorithm 2 and line (5) in the randQB.b algorithm can also be proved, although it is less obvious. With the matrix-vector multiplications lumped into two matrix-matrix multiplications, more efficiency benefit from the BLAS-3 operation is expected. Algorithm 2 also exhibits the *pass-efficient* property, like the original algorithm in [2].

Algorithm 2 A basic pass-efficient algorithm for auto-rank QB factorization

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, desired accuracy tolerance ε , block size b .

Output: \mathbf{Q} , \mathbf{B} , s.t. $\|\mathbf{A} - \mathbf{QB}\| < \varepsilon$.

```

1:  $\mathbf{Q} = []$ ;  $\mathbf{B} = []$ ;
2:  $\mathbf{\Omega} = \text{randn}(n, \tilde{l})$ , where  $\tilde{l}$  is a sufficiently large number.
3:  $\mathbf{G} = \mathbf{A}\mathbf{\Omega}$ 
4:  $\mathbf{H} = \mathbf{A}^T \mathbf{G}$ 
5:  $E = \|\mathbf{A}\|^2$ 
6: for  $i = 1, 2, 3, \dots$  do
7:    $\mathbf{\Omega}_i = \mathbf{\Omega}(:, (i-1)b + 1 : ib)$ 
8:    $\mathbf{Y}_i = \mathbf{G}(:, (i-1)b + 1 : ib) - \mathbf{Q}(\mathbf{B}\mathbf{\Omega}_i)$ 
9:    $[\mathbf{Q}_i, \mathbf{R}_i] = \text{qr}(\mathbf{Y}_i)$ 
10:   $\mathbf{B}_i = \mathbf{R}_i^{-T}(\mathbf{H}(:, (i-1)b + 1 : ib)^T - \mathbf{\Omega}_i^T \mathbf{B}^T \mathbf{B})$ 
11:   $\mathbf{Q} = [\mathbf{Q}, \mathbf{Q}_i]$ 
12:   $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_i \end{bmatrix}$ 
13:   $E = E - \|\mathbf{B}_i\|^2$ 
14:  if  $\sqrt{E} < \varepsilon$  then stop
15: end for
```

PROPOSITION 2. $\forall i$, $\mathbf{Q}^{(i)}$ in Algorithm 2 is orthonormal and the corresponding $\mathbf{B}^{(i)} = \left(\mathbf{Q}^{(i)}\right)^T \mathbf{A}$.

Proof. We prove it via induction. In base case, $\mathbf{Q}^{(1)} = \mathbf{Q}_1$ is orthonormal because of line 9. It also ensures that \mathbf{Q}_i is orthonormal, and

$$(24) \quad \mathbf{Q}_i \mathbf{R}_i = \mathbf{Y}_i.$$

So,

$$(25) \quad \mathbf{B}^{(1)} = \mathbf{B}_1 = \mathbf{R}_1^{-T} \boldsymbol{\Omega}_1^T \mathbf{A}^T \mathbf{A} = (\mathbf{A} \boldsymbol{\Omega}_1 \mathbf{R}_1^{-1})^T \mathbf{A} = (\mathbf{Y}_1 \mathbf{R}_1^{-1})^T \mathbf{A} = \left(\mathbf{Q}^{(1)} \right)^T \mathbf{A},$$

where the last two equalities are due to line 8 in Algorithm 2 and (24), respectively.

Now, suppose the proposition holds for the i -th iteration. We need to prove $\mathbf{Q}^{(i+1)}$ is orthonormal and $\mathbf{B}^{(i+1)} = \left(\mathbf{Q}^{(i+1)} \right)^T \mathbf{A}$. We first check if \mathbf{Q}_{i+1} is perpendicular to $\mathbf{Q}^{(i)}$.

$$(26) \quad \begin{aligned} \mathbf{Q}_{i+1}^T \mathbf{Q}^{(i)} &= \left(\left(\mathbf{A} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)} \right) \boldsymbol{\Omega}_{i+1} \mathbf{R}_{i+1}^{-1} \right)^T \mathbf{Q}^{(i)} \\ &= \left(\left(\mathbf{A} - \mathbf{Q}^{(i)} \left(\mathbf{Q}^{(i)} \right)^T \mathbf{A} \right) \boldsymbol{\Omega}_{i+1} \mathbf{R}_{i+1}^{-1} \right)^T \mathbf{Q}^{(i)} \\ &= \left((\mathbf{I} - \mathbf{P}_{\mathbf{Q}^{(i)}}) \mathbf{A} \boldsymbol{\Omega}_{i+1} \mathbf{R}_{i+1}^{-1} \right)^T \mathbf{Q}^{(i)} \\ &= \left(\mathbf{A} \boldsymbol{\Omega}_{i+1} \mathbf{R}_{i+1}^{-1} \right)^T (\mathbf{I} - \mathbf{P}_{\mathbf{Q}^{(i)}})^T \mathbf{Q}^{(i)} \\ &= \left(\mathbf{A} \boldsymbol{\Omega}_{i+1} \mathbf{R}_{i+1}^{-1} \right)^T (\mathbf{Q}^{(i)} - \mathbf{P}_{\mathbf{Q}^{(i)}} \mathbf{Q}^{(i)}) \\ &= \mathbf{O}. \end{aligned}$$

The last equality of (26) is due to Lemma 1. Note that $\mathbf{P}_{\mathbf{Q}^{(i)}}$ is an orthogonal projector. Eq. (26) guarantees that $\mathbf{Q}^{(i+1)}$ is an orthonormal matrix. Then,

$$(27) \quad \begin{aligned} \mathbf{B}_{i+1} &= \mathbf{R}_{i+1}^{-T} \boldsymbol{\Omega}_{i+1}^T \left(\mathbf{A}^T \mathbf{A} - \mathbf{B}^{(i)T} \mathbf{B}^{(i)} \right) \\ &\stackrel{1}{=} \mathbf{R}_{i+1}^{-T} \boldsymbol{\Omega}_{i+1}^T \left(\mathbf{A} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)} \right)^T \left(\mathbf{A} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)} \right) \\ &= \left(\mathbf{A} \boldsymbol{\Omega}_{i+1} \mathbf{R}_{i+1}^{-1} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)} \boldsymbol{\Omega}_{i+1} \mathbf{R}_{i+1}^{-1} \right)^T \left(\mathbf{A} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)} \right) \\ &= \left(\mathbf{Y}_{i+1} \mathbf{R}_{i+1}^{-1} \right)^T \left(\mathbf{A} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)} \right) \\ &= \mathbf{Q}_{i+1}^T \left(\mathbf{A} - \mathbf{Q}^{(i)} \mathbf{B}^{(i)} \right) \\ &\stackrel{2}{=} \mathbf{Q}_{i+1}^T \mathbf{A}, \end{aligned}$$

where equality 1 holds due to Lemma 2, and equality 2 follows from (26). Therefore, we see that $\mathbf{B}^{(i+1)} = \left(\mathbf{Q}^{(i+1)} \right)^T \mathbf{A}$, based on the induction hypothesis and lines 11 and 12 in Algorithm 2. This ends the proof. \square

For the stopping criterion, it is just the same as Algorithm 1.

4.2. The version with re-orthogonalization. In reality, the loss of orthonormality among the columns in $\{\mathbf{Q}_1, \mathbf{Q}_2, \dots\}$ occurs due to the accumulation of round-off errors. This further affects the correctness of some statements in Algorithm 2, and increases the error of its output. To fix this problem, we explicitly reproject \mathbf{Q}_i away from the span of the previously computed basis vectors, just as what's done in [1]. Then, we modify the formula for matrix \mathbf{B}_i to incorporate the modified \mathbf{Q}_i .

The re-orthogonalization can be expressed as

$$(28) \quad \tilde{\mathbf{Q}}_i \tilde{\mathbf{R}}_i = \mathbf{Q}_i - \mathbf{Q}^{(i-1)} \left(\mathbf{Q}^{(i-1)} \right)^T \mathbf{Q}_i,$$

where $\tilde{\mathbf{Q}}_i \neq \mathbf{Q}_i$ and $\tilde{\mathbf{R}}_i \neq \mathbf{I}$ due to round-off error. And, $\tilde{\mathbf{Q}}_i$ is better orthogonal to the previously generated $\{\mathbf{Q}_1, \mathbf{Q}_2, \dots, \mathbf{Q}_{i-1}\}$ than \mathbf{Q}_i . Now, we derive a formula for calculating \mathbf{B}_i which does not involve \mathbf{A} explicitly. Based on (24),

$$(29) \quad \tilde{\mathbf{Q}}_i = \left(\mathbf{I} - \mathbf{Q}^{(i-1)} \left(\mathbf{Q}^{(i-1)} \right)^T \right) \mathbf{Y}_i \mathbf{R}_i^{-1} \tilde{\mathbf{R}}_i^{-1}.$$

(30)

$$\begin{aligned} \tilde{\mathbf{B}}_i &= \tilde{\mathbf{Q}}_i^T \mathbf{A} \\ &= (\tilde{\mathbf{R}}_i \mathbf{R}_i)^{-T} \mathbf{Y}_i^T \left(\mathbf{I} - \mathbf{Q}^{(i-1)} \left(\mathbf{Q}^{(i-1)} \right)^T \right) \mathbf{A} \\ &= (\tilde{\mathbf{R}}_i \mathbf{R}_i)^{-T} \left(\mathbf{\Omega}_i^T \mathbf{A}^T - \mathbf{\Omega}_i^T \left(\mathbf{B}^{(i-1)} \right)^T \left(\mathbf{Q}^{(i-1)} \right)^T \right) \left(\mathbf{A} - \mathbf{Q}^{(i-1)} \left(\mathbf{Q}^{(i-1)} \right)^T \mathbf{A} \right) \\ &\approx (\tilde{\mathbf{R}}_i \mathbf{R}_i)^{-T} \left(\mathbf{H}_i^T - \mathbf{G}_i^T \mathbf{Q}^{(i-1)} \mathbf{B}^{(i-1)} - \mathbf{\Omega}_i^T \left(\mathbf{B}^{(i-1)} \right)^T \mathbf{B}^{(i-1)} + \mathbf{\Omega}_i^T \left(\mathbf{B}^{(i-1)} \right)^T \left(\mathbf{Q}^{(i-1)} \right)^T \mathbf{Q}^{(i-1)} \mathbf{B}^{(i-1)} \right) \\ &= (\tilde{\mathbf{R}}_i \mathbf{R}_i)^{-T} \left(\mathbf{H}_i^T - \mathbf{Y}_i^T \mathbf{Q}^{(i-1)} \mathbf{B}^{(i-1)} - \mathbf{\Omega}_i^T \left(\mathbf{B}^{(i-1)} \right)^T \mathbf{B}^{(i-1)} \right). \end{aligned}$$

In the deduction, we use the formula for \mathbf{Y}_i shown as line 8 in Algorithm 2. And, we try our best to less utilize the orthogonal property of $\mathbf{Q}^{(i-1)}$ and the equality $\mathbf{B}^{(i-1)} = \left(\mathbf{Q}^{(i-1)} \right)^T \mathbf{A}$, which may not hold in the floating-point computation.

Based on (28) and (30), we can derive the version with re-orthogonalization for Algorithm 2. We just need to replace step 10 with the following steps:

$$\begin{aligned} 10: & \quad [\mathbf{Q}_i, \tilde{\mathbf{R}}_i] = \text{qr}(\mathbf{Q}_i - \mathbf{Q}(\mathbf{Q}^T \mathbf{Q}_i)) \\ 10': & \quad \mathbf{R}_i = \tilde{\mathbf{R}}_i \mathbf{R}_i \\ 10'': & \quad \mathbf{B}_i = \mathbf{R}_i^{-T} (\mathbf{H}(:, (i-1)b+1 : ib)^T - \mathbf{Y}_i^T \mathbf{Q} \mathbf{B} - \mathbf{\Omega}_i^T \mathbf{B}^T \mathbf{B}) \end{aligned}$$

Here, \mathbf{Q}_i and \mathbf{B}_i are overwritten to stand for $\tilde{\mathbf{Q}}_i$ and $\tilde{\mathbf{B}}_i$.

This algorithm with fewer passes over \mathbf{A} is denoted by `randQB_FP`. Actually, it is a single-pass algorithm, because steps 3 and 4 can be implemented with only single pass over \mathbf{A} . In contrast, the `randQB_b` and `randQB_EI` algorithms need $3l/b$ and $2l/b$ passes over \mathbf{A} , respectively.

The re-orthogonalization induces some extra computation to Algorithm 2. In the i -th iteration, it is of $O(3ib^2m + ib^2n)$ flops. Summing them for all iterations, it becomes $O(3l^2m/2 + l^2n/2)$, where l is the number of columns in the resulting \mathbf{Q} . This is much less than the cost of $O(mn\tilde{l})$ flops for executing step 3 or 4 (assuming a dense \mathbf{A}). Here, we ignore the computations with respect to \mathbf{R}_i and $\tilde{\mathbf{R}}_i$, as they are in a small size b . The runtime of the `randQB_FP` algorithm is roughly:

$$(31) \quad \begin{aligned} T_{\text{randQB_FP}} &\sim 2C_{mm}mnl + 4C_{mm}(m+n)b^2 \sum_{i=1}^{t-1} i + 2C_{qr}mb^2t \\ &\sim 2C_{mm}mnl + 2C_{mm}(m+n)l^2 + \frac{2}{t}C_{qr}ml^2, \end{aligned}$$

where t satisfies $l = tb$. For simplicity, we do not differentiate l and \tilde{l} , although the latter is often larger than the former.

Compared with `TrandQB_EI` for Algorithm 1, the `randQB_FP` algorithm has slightly larger flop count. However, since the multiplication with \mathbf{A} is lumped together, its actual runtime may be shorter especially on a parallel computing platform. While

<hr/> function $[\mathbf{Q}, \mathbf{B}] = \text{randQB_p}(\mathbf{A}, \ell, P)$ <hr/>	<hr/> function $[\mathbf{Q}, \mathbf{B}] = \text{randQB_pb}(\mathbf{A}, \varepsilon, P, b)$ <hr/>
(1) $\mathbf{\Omega} = \text{randn}(n, \ell).$	(1) for $i = 1, 2, 3, \dots$
(2) $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{\Omega}).$	(2) $\mathbf{\Omega}_i = \text{randn}(n, b).$
(3) for $j = 1 : P$	(3) $\mathbf{Q}_i = \text{orth}(\mathbf{A}\mathbf{\Omega}_i).$
(4) $\mathbf{Q} = \text{orth}(\mathbf{A}^* \mathbf{Q}).$	(4) for $j = 1 : P$
(5) $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{Q}).$	(5) $\mathbf{Q}_i = \text{orth}(\mathbf{A}^* \mathbf{Q}_i).$
(6) end for	(6) $\mathbf{Q}_i = \text{orth}(\mathbf{A}\mathbf{Q}_i).$
(7) $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ <hr/>	(7) end for
	(8) $\mathbf{Q}_i = \text{orth}(\mathbf{Q}_i - \sum_{j=1}^{i-1} \mathbf{Q}_j \mathbf{Q}_j^* \mathbf{Q}_i)$
	(9) $\mathbf{B}_i = \mathbf{Q}_i^* \mathbf{A}$
	(10) $\mathbf{A} = \mathbf{A} - \mathbf{Q}_i \mathbf{B}_i$
	(11) if $\ \mathbf{A}\ < \varepsilon$ then stop
	(12) end while
	(13) Set $\mathbf{Q} = [\mathbf{Q}_1 \dots \mathbf{Q}_i]$ and $\mathbf{B} = [\mathbf{B}_1^* \dots \mathbf{B}_i^*]^*.$ <hr/>

(a) The randQB_p algorithm

(b) The randQB_pb algorithm

FIG. 4. The basic randQB algorithm and the blocked randQB algorithm using the power scheme (from [1]). P is an integer “power” parameter, b is the block size, and “*” denotes matrix transpose.

comparing the randQB_FP and the blocked randQB algorithm, i.e (31) vs. (22), we see that the former costs less flops. Same as the randQB_EI algorithm, the randQB_FP algorithm also adapts to a sparse matrix \mathbf{A} . Besides, the randQB_FP algorithm owns the unique single-pass merit which is favorable for the extremely large \mathbf{A} .

5. The power scheme for improving accuracy. Based on the theoretic error bound of the randomized QB factorization [2], the error $\|\mathbf{A} - \mathbf{QB}\|$ depends on the quantity $(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2)^{1/2}$ and a scalar factor in front of it. This implies that the error could be large if the “tail” singular values have substantial weight, e.g. those of a matrix whose singular values decay slowly. This problem becomes particularly acute for large matrices. Fortunately, it can be resolved by using a so called *power scheme* [11]. Suppose P is an integer. The power scheme is inspired by the fact that matrix $(\mathbf{A}\mathbf{A}^T)^P \mathbf{A}$ has exactly the same left/right singular vectors as \mathbf{A} , but its singular values are σ_j^{2P+1} . Due to this power operation, the relative weights of the tail singular values of $(\mathbf{A}\mathbf{A}^T)^P \mathbf{A}$ can be much smaller than that of \mathbf{A} . So, performing the randomized QB factorization on $(\mathbf{A}\mathbf{A}^T)^P \mathbf{A}$ may achieve better accuracy, while producing the same leading left singular vectors of \mathbf{A} (i.e. the orthonormal basis for $\text{range}(\mathbf{A})$). More theoretic analysis reveals that with the power scheme the scalar factor in the error bound is largely reduced, thus resulting in more accurate approximation. For the details, please refer to [2, Sec. 10.4].

To implement the power scheme, conceptually we just need to replace matrix \mathbf{A} in the algorithm with $(\mathbf{A}\mathbf{A}^T)^P \mathbf{A}$. The basic randQB procedure and the blocked randQB algorithm with the power scheme are shown in Figure 4. It should be pointed out that in floating-point computation, substantial loss of accuracy occurs whenever the singular values have a large variation. If ϵ_{mach} denotes the machine precision, any singular components smaller than $\sigma_1 \epsilon_{mach}^{1/(2P+1)}$ will be lost. This problem can be resolved by orthonormalizing the “sample matrix” between each application of \mathbf{A} and \mathbf{A}^T . This explains the orthonormalization steps in Figure 4.

Similarly, the randQB_EI algorithm can be extended to incorporate the power scheme. The algorithm is presented as Algorithm 3. Compared with Algorithm 1, we only add lines 6 through 9, where $\mathbf{A} - \mathbf{QB}$ replaces the original \mathbf{A} .

We combine the power scheme and the randQB_FP algorithm to obtain Algorithm 4. Because matrix \mathbf{A} only appears outside the loop, we perform the applications of \mathbf{A}

Algorithm 3 Accuracy-enhanced randQB-EI using the power scheme

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, desired accuracy tolerance ε , block size b , power P .

Output: \mathbf{Q}, \mathbf{B} , s.t. $\|\mathbf{A} - \mathbf{Q}\mathbf{B}\| < \varepsilon$.

```

1:  $\mathbf{Q} = []$ ;  $\mathbf{B} = []$ ;
2:  $E = \|\mathbf{A}\|^2$ 
3: for  $i = 1, 2, 3, \dots$  do
4:    $\mathbf{\Omega}_i = \text{randn}(n, b)$ 
5:    $\mathbf{Q}_i = \text{orth}(\mathbf{A}\mathbf{\Omega}_i - \mathbf{Q}(\mathbf{B}\mathbf{\Omega}_i))$ 
6:   for  $j = 1 : P$  do
7:      $\mathbf{Q}_i = \text{orth}(\mathbf{A}^T \mathbf{Q}_i - \mathbf{B}^T (\mathbf{Q}^T \mathbf{Q}_i))$ 
8:      $\mathbf{Q}_i = \text{orth}(\mathbf{A} \mathbf{Q}_i - \mathbf{Q}(\mathbf{B} \mathbf{Q}_i))$ 
9:   end for
10:   $\mathbf{Q}_i = \text{orth}(\mathbf{Q}_i - \mathbf{Q}(\mathbf{Q}^T \mathbf{Q}_i))$ 
11:   $\mathbf{B}_i = \mathbf{Q}_i^T \mathbf{A} - \mathbf{Q}_i^T \mathbf{Q} \mathbf{B}$ 
12:   $\mathbf{Q} = [\mathbf{Q}, \mathbf{Q}_i]$ 
13:   $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_i \end{bmatrix}$ 
14:   $E = E - \|\mathbf{B}_i\|^2$ 
15:  if  $\sqrt{E} < \varepsilon$  then stop
16: end for
```

and \mathbf{A}^T and the orthonormalization steps there. An important point is that we must ensure that $\mathbf{G} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{H} = \mathbf{A}^T \mathbf{G}$ for their usage in the loop. Our experiments reveal that the orthonormalization cannot be omitted. Otherwise, the accuracy of the QB factorization will be largely degraded.

The flop counts of Algorithm 3 and 4 are roughly (for an $m \times n$ dense matrix \mathbf{A}):

(32)

$$\begin{aligned}
T_{\text{randQB-EI-P}} &\sim T_{\text{randQB-EI}} + 2C_{mm}Pmnl + 2C_{mm}P(m+n)b^2 \sum_{i=0}^{t-1} i + C_{qr}P(m+n)b^2t \\
&\sim T_{\text{randQB-EI}} + 2PC_{mm}mnl + PC_{mm}(m+n)l^2 + \frac{1}{t}PC_{qr}(m+n)l^2 \\
&\sim C_{mm}(2+2P)mnl .
\end{aligned}$$

(33)

$$\begin{aligned}
T_{\text{randQB-FP-P}} &\sim T_{\text{randQB-FP}} + 2C_{mm}Pmnl + C_{qr}P(m+n)\tilde{l}^2 \\
&\sim C_{mm}(2+2P)mnl .
\end{aligned}$$

Again, we ignore the difference between l and \tilde{l} . Since l is much smaller than m or n in a low-rank factorization, we only keep the dominant first item.

The computational cost of randQB-p and randQB-pb are [1]:

(34)

$$T_{\text{randQB-p}} \sim C_{mm}(2+2P)mnl + C_{qr}(1+2P)ml^2 ,$$

(35)

$$T_{\text{randQB-pb}} \sim C_{mm}(3+2P)mnl + C_{mm}ml^2 + \frac{1}{t}C_{qr}(2+2P)ml^2 .$$

Comparing these formulas, we see that the proposed algorithms cost less time than the counterpart of the blocked randQB algorithm. They are also suitable for sparse

Algorithm 4 Accuracy-enhanced randQB_FP using the power scheme

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, desired accuracy tolerance ε , block size b , power P .

Output: \mathbf{Q}, \mathbf{B} , s.t. $\|\mathbf{A} - \mathbf{QB}\| \leq \varepsilon$.

```
1:  $\mathbf{Q} = []$ ;  $\mathbf{B} = []$ ;
2:  $\mathbf{\Omega} = \text{randn}(n, \tilde{l})$ , where  $\tilde{l}$  is a sufficiently large number.
3: for  $i = 1 : P$  do
4:    $\mathbf{G} = \text{orth}(\mathbf{A}\mathbf{\Omega})$ 
5:    $\mathbf{\Omega} = \text{orth}(\mathbf{A}^T \mathbf{G})$ 
6: end for
7:  $\mathbf{G} = \mathbf{A}\mathbf{\Omega}$ 
8:  $\mathbf{H} = \mathbf{A}^T \mathbf{G}$ 
9:  $E = \|\mathbf{A}\|^2$ 
10: for  $i = 1, 2, 3, \dots$  do
11:    $\mathbf{\Omega}_i = \mathbf{\Omega}(:, (i-1)b + 1 : ib)$ 
12:    $\mathbf{Y}_i = \mathbf{G}(:, (i-1)b + 1 : ib) - \mathbf{Q}(\mathbf{B}\mathbf{\Omega}_i)$ 
13:    $[\mathbf{Q}_i, \mathbf{R}_i] = \text{qr}(\mathbf{Y}_i)$ 
14:    $[\mathbf{Q}_i, \tilde{\mathbf{R}}_i] = \text{qr}(\mathbf{Q}_i - \mathbf{Q}(\mathbf{Q}^T \mathbf{Q}_i))$ 
15:    $\mathbf{R}_i = \tilde{\mathbf{R}}_i \mathbf{R}_i$ 
16:    $\mathbf{B}_i = \mathbf{R}_i^{-T}(\mathbf{H}(:, (i-1)b + 1 : ib)^T - \mathbf{Y}_i^T \mathbf{QB} - \mathbf{\Omega}_i^T \mathbf{B}^T \mathbf{B})$ 
17:    $\mathbf{Q} = [\mathbf{Q}, \mathbf{Q}_i]$ 
18:    $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_i \end{bmatrix}$ 
19:    $E = E - \|\mathbf{B}_i\|^2$ 
20:   if  $\sqrt{E} < \varepsilon$  then stop
21: end for
```

matrix, and cost less memory. For the randQB_FP with power scheme, the number of passes over matrix \mathbf{A} increases to $2P + 1$. In most scenarios, where $P = 1$ or 2 brings sufficient accuracy, the accuracy-enhanced randQB_FP is still pass-efficient.

6. Discussion on limitations of the algorithms. The proposed techniques are inspired by the blocked randQB scheme in [1]. They employ an economic error indicator (i.e. E) to remove the costly explicit computation of the error matrix. However, this error indicator causes a limitation to the algorithms. Based on Lemma 3, we calculate $E = \|\mathbf{A}\|^2 - \|\mathbf{B}\|^2$, and then \sqrt{E} indicates the approximation error of \mathbf{QB} . Let's consider an extreme scenario, where E is *theoretically* as small as $\epsilon_{mach}\|\mathbf{A}\|^2$. This means $\|\mathbf{A}\|^2$ and $\|\mathbf{B}\|^2$ have the same decimal digits except the last digit. Because the last decimal digit in the floating-number arithmetic is just a matter of error, the difference of them, E , does not include any useful (accurate) information. This is the phenomena of “cancellation”. So, we cannot expect accurately obtaining an E smaller than $\epsilon_{mach}\|\mathbf{A}\|^2$. Sometime it could be a negative value, causing calculating \sqrt{E} a problem. This means if

$$(36) \quad \|\mathbf{A} - \mathbf{QB}\|^2 < \epsilon_{mach}\|\mathbf{A}\|^2, \text{ i.e. } \frac{\|\mathbf{A} - \mathbf{QB}\|}{\|\mathbf{A}\|} < \sqrt{\epsilon_{mach}} \approx 10^{-8} \text{ in double-precision,}$$

E becomes meaningless. Therefore, the proposed algorithms cannot be used for the problems where we want to pursue a relative error below 10^{-8} . This rarely happens as most actual low-rank approximation problems do not request such high accuracy.

Another problem of the randQB_FP algorithm is due to the inaccuracy in calculating \mathbf{B}_i , c.f. (30). With the iteration steps increase for high-accuracy approximation,

the algorithm might not guarantee the orthogonality of \mathbf{Q} matrix, and prevents approximation error from continuously decreasing. But this hardly occurs before the limitation of the error indicator emerges. That is, for the applications where we pursue an approximation with moderate relative error, say no less than 10^{-7} , these numerical drawbacks are negligible. In the following section, we will show numerical results to validate the usage and effectiveness of the proposed algorithms.

7. Numerical results. In this section we compare our algorithms against several existing algorithms in terms of execution time, memory usage and accuracy. All experiments are carried out on a Linux server with two 12-core Intel Xeon E5-2630 CPUs (2.30 GHz), and 32GB of RAM. For comparison of speed, the proposed algorithms have been implemented in C based on the codes shared by the authors of [1, 12]. The program is coded with OpenMP derivatives, and compiled with the Intel ICC compiler with MKL libraries [13], to take full advantage of the multi-core CPUs. The QR factorization and other basic linear algebra operations are implemented through LAPACK routines which are automatically executed in parallel.

7.1. Comparison of speed. We compute the QB factorization (with rank l) of a given $n \times n$ matrix \mathbf{A} . Notice the singular value distribution of matrix is immaterial for a runtime comparison. Four sets of techniques are compared:

- The basic randQB procedure, i.e. the algorithm in Figure 4(a);
- The blocked randQB algorithm [1];
- The randQB_EI algorithm with power scheme, i.e. Algorithm 3 in Section 5;
- The randQB_FP algorithm with power scheme, i.e. Algorithm 4 in Section 5.

The block size is $b = 20$ for the blocked randQB, randQB_EI and randQB_FP algorithms. We did not test the partial QR factorization, because the experiments in [1] showed that the block randQB algorithm is nearly 10X faster than it.

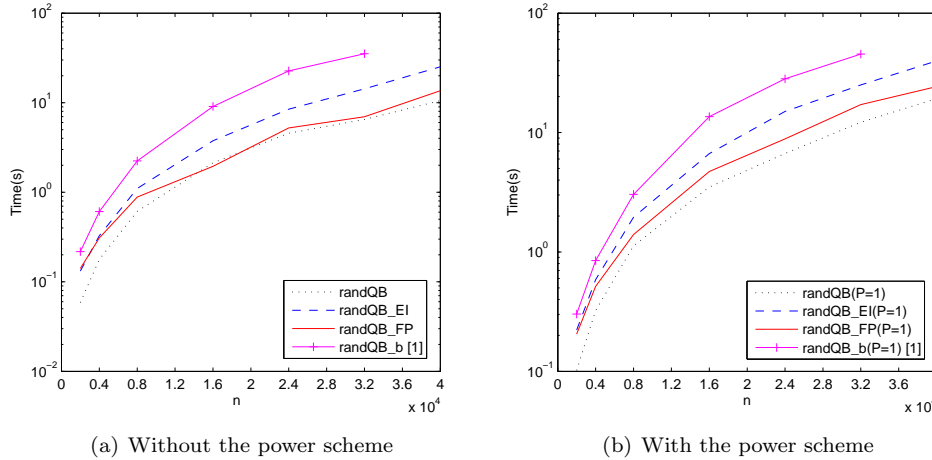


FIG. 5. Runtime of the algorithms for dense matrices ($l = 200$).

In the first experiment we test the algorithms on dense matrices of varying size. n ranges from 2000 to 40000. The value of l is fixed to 200. The results are shown in Figure 5 for the situations without and with the power scheme. The data of the blocked randQB algorithm for the matrix with $n = 40000$ are not shown due to unreasonably long runtime of the program from [12]. From the results, we see the randQB_EI and randQB_FP algorithms are more than 2X and 4X faster than the blocked randQB algorithm, respectively. When the power scheme is imposed, the

acceleration ratios decrease a little bit, but is still about 2 or more. The basic randQB procedure has the fastest computational speed, while the randQB_FP algorithm is faster than the randQB_EI algorithm. Sometimes, the randQB_FP is even faster than the basic randQB procedure.

We do not show the runtime results regarding to the power scheme with $P = 2$, because it has almost the same accuracy as the power scheme with $P = 1$, despite costing more computing time. This will be illustrated in the next subsection.

The memory costs for some large matrices are listed in Table 1. For the randQB, randQB_EI and randQB_FP algorithms, the memory cost is mainly that for storing matrix \mathbf{A} . For the randQB_b algorithm, it is mainly for the error matrix (assuming \mathbf{A} is overwritten), and the product of \mathbf{QB} . So, the proposed algorithms consume half of that used by the blocked randQB algorithm. If we want to keep the original \mathbf{A} , the proposed algorithms cost only 1/3 of the memory used by the algorithm in [1].

TABLE 1
The memory usage of the randomized QB-factorization algorithms for dense matrices ($l = 200$)

n	randQB	randQB_EI	randQB_FP	randQB_b [1]
16000	2.25 GB	2.0 GB	2.1 GB	3.9 GB
24000	4.5 GB	4.5 GB	4.6 GB	8.4 GB
32000	8.0 GB	7.9 GB	8.0 GB	15 GB
40000	12 GB	12 GB	12 GB	–
n	randQB($P=1$)	randQB_EI($P=1$)	randQB_FP($P=1$)	randQB_b($P=1$) [1]
16000	2.1 GB	2.0 GB	2.1 GB	3.9 GB
24000	4.5 GB	4.5 GB	4.6 GB	8.4 GB
32000	7.9 GB	7.8 GB	7.9 GB	15 GB
40000	12 GB	12 GB	12 GB	–

The second experiment is about the algorithms’ efficiency for sparse matrices. We generate sparse matrices with roughly 0.3% non-zero elements. They are stored in CSR (compressed sparse row) format [14]. The runtimes of the algorithms are shown in Figure 6. The data of the randQB_b algorithm for the matrices with $n \geq 40000$ are not available due to their unreasonably long runtime. In contrast, it only takes a couple of seconds for the other algorithms to process the largest matrix with $n = 48000$. We see that the proposed algorithms take remarkable benefit from the sparsity, while the blocked randQB algorithm cannot. The speedup ratios of the former to the latter increase as the matrix size increases. For $n = 32000$, the randQB_EI and randQB_FP algorithms are more than 20X and 10X faster than the randQB_b algorithm, respectively. Different from the situation when we test dense matrices, the randQB_EI algorithm becomes faster than randQB_FP. This implies that lumping the multiplications of a sparse matrix all together brings less benefit than doing that for a dense matrix. And, randQB_EI could run faster than the basic randQB procedure, because the “orth” operator in the latter has higher cost, and the orthogonalization steps dominate the total runtime in the treatment of sparse matrix. Another interesting phenomenon is that if we instead store the sparse matrix with the COO (coordinate) format the randQB_FP algorithm runs up to 30% faster. A possible explanation could be that the COO format is more adaptive to the parallel computing.

The memory cost of these algorithms are listed in Table 2. From the table we see more prominent memory saving of the proposed methods.

Lastly, we fix the size of matrix ($n = 8000$), but vary the value of rank l . The trends of the runtime are plotted in Figure 7. It shows that our algorithms are about 2X faster than randQB_b. If the power scheme is imposed, the speedup ratio of randQB_EI to randQB_b decreases with the increase of l , while that of randQB_FP

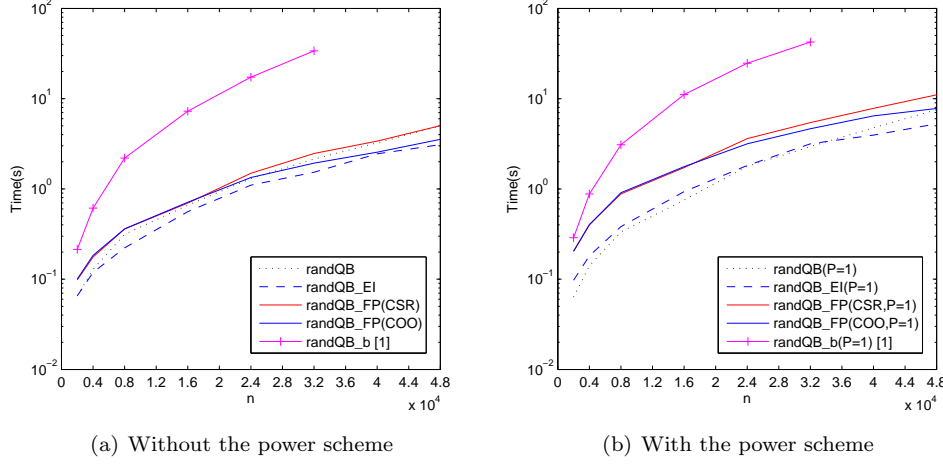


FIG. 6. Runtime of the algorithms for sparse matrices ($l = 200$).

TABLE 2
The memory usage of the randomized QB-factorization algorithms for sparse matrices ($l = 200$)

n	randQB	randQB_EI	randQB_FP	randQB_b[1]
16000	230 MB	149 MB	213 MB	3.9 GB
24000	228 MB	231 MB	314 MB	8.4 GB
32000	268 MB	308 MB	420 MB	15 GB
40000	303 MB	409 MB	490 MB	—
48000	385 MB	426 MB	573 MB	—
n	randQB($P=1$)	randQB_EI($P=1$)	randQB_FP($P=1$)	randQB_b($P=1$) [1]
16000	220 MB	153 MB	316 MB	3.9 GB
24000	228 MB	220 MB	445 MB	8.4 GB
32000	282 MB	322 MB	501 MB	15 GB
40000	305 MB	411 MB	590 MB	—
48000	389 MB	478 MB	664 MB	—

keeps to a value larger than 2.

7.2. Comparison of accuracy. In this subsection, we investigate the accuracy of the proposed randomized schemes. Three kinds of test matrices are tested standing for different distributions of singular values:

- **Matrix 1 (slow decay):** $\mathbf{A}_1 = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$, where \mathbf{U} and \mathbf{V} are randomly drawn matrices with orthonormal columns, and the diagonal matrix $\mathbf{\Sigma}$ has diagonal elements $\sigma_{jj} = 1/j^2$.
- **Matrix 2 (fast decay):** \mathbf{A}_2 is formed just like \mathbf{A}_1 , but the diagonal elements of $\mathbf{\Sigma}$ is given by $\sigma_{jj} = e^{-j/7}$. It reflects a fast decay of singular values.
- **Matrix 3 (S-shape decay):** \mathbf{A}_3 is built in the same manner as \mathbf{A}_1 and \mathbf{A}_1 , but the diagonal elements of $\mathbf{\Sigma}$ are given by $\sigma_{jj} = 0.0001 + (1 + e^{j-30})^{-1}$. It makes the singular values first hover around 1, then decay rapidly, and finally level out at about 0.0001.

We compare the proposed techniques with the blocked randQB scheme [1], and the optimal technique based on the truncated SVD.

For each case, we generate a 2000×2000 matrix, for which we compare the errors of the above techniques for computing rank- l approximation. The results are shown in Figure 8. From it we see that the proposed techniques have just the same accuracy as the blocked randQB technique. They are mathematically equivalent after all.

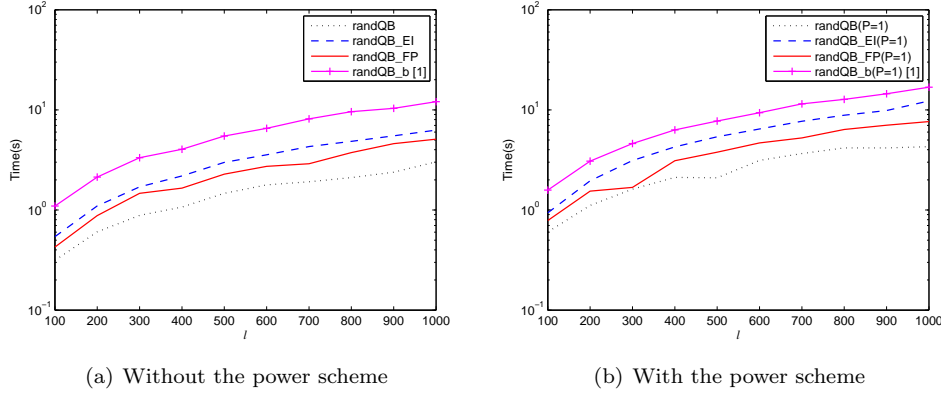


FIG. 7. Runtime of the algorithms for a dense matrix ($n = 8000$) with varying rank.

If we use the power scheme, even with as low of a power as $P = 1$, the `randQB_EI` and `randQB_FP` algorithms produce essentially the near-optimal results, like the SVD based technique. And, the power schemes with $P = 1$ and $P = 2$ produce indistinguishable results even for the matrix with slow decay of singular values. The data shown in Figure 8 refers to a single instantiation of the randomized algorithm; but they have already shown sufficient accuracy.

The validity of the error indicator proposed in Section 3 is critical for the effectiveness of the `randQB_EI` and `randQB_FP` algorithms. To evaluate its accuracy, we've tested Matrix 2, and drawn the relative errors of approximation and the values of error indicator as l increases in Figure 9. From the figure we see that, if the actual error is no less than about 5×10^{-7} the indicator in `randQB_EI` or `randQB_FP` algorithm well matches the actual error. When the error becomes smaller (with the rank increases), the error inductor stagnates, failing to reflect actual error. This is explained in Section 6, and the experiment here just validates that analysis.

7.3. Performance for the auto-rank problem. The proposed `randQB_EI` and `randQB_FP` algorithms are aimed at solving the auto-rank problem. The usually-used accuracy tolerance is $\|\mathbf{A} - \mathbf{QB}\| < \delta \|\mathbf{A}\|$. The optimal solution is the factorization with the smallest rank, because smaller rank corresponds to smaller amount of subsequent computation based on the low-rank factorization. Due to the property of the truncated SVD, we can first make SVD of the input matrix \mathbf{A} , and then check $(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2)^{1/2}$, where σ_j is \mathbf{A} 's j -th singular value, to determine the smallest rank k satisfying the accuracy tolerance. This optimal rank comes with the large computational cost of a full SVD.

In this subsection we compare the proposed algorithms with the SVD based method producing the optimal rank, and the *adaptive randomized range finder* (Algorithm 4.2 in [2]). The experiments are carried out with Matlab version 2012a and the algorithms' Matlab implementations, on the aforementioned Linux server. The built-in commands like "svd", "qr", etc. are employed. They naturally take advantage of the parallel computing. In the `randQB_FP` algorithm, we set $\bar{l} = 50b$. And, the row-by-row scheme mentioned in Section 3 is implemented into the algorithms.

We first construct the three kinds of matrices in last subsection, each in 8000×8000 size. Setting different tolerances for the relative error of approximation, we test the four algorithms. Their results are shown in Table 3. For `randQB_EI` and `randQB_FP`, the power scheme with $P = 1$ is used. The block size is set to $b = 10$ in all tests,

except the last one for which $b = 40$. In Table 3, “ δ ” stands for the threshold for relative error, and “error” means the relative error of the produced QB factorization. From the results we see that the QB factorizations outputted by `randQB_EI` and `randQB_FP` algorithms all satisfy the set accuracy demands. And, the obtained ranks are very close to the the optimal ranks obtained with the SVD based approach. As for the runtime, the proposed algorithms are usually several tens times faster than the SVD operation. Notice that our Matlab programs are less optimized than the built-in `svd` command. So, more speedup is expected for the implementation based on C. Although the adaptive range finder is built on a theory with spectral norm of matrix, in our experiment with each relative-error tolerance it always produces a QB factorization satisfying the accuracy demand in Frobenius norm. However, from Table 3 we see that the adaptive range finder largely overestimates the rank.

TABLE 3
The results of auto-rank problems based on the three test matrices

Matrix	δ	randQB_EI			randQB_FP		truncated SVD		RangeFinder [2]	
		rank	time(s)	error	rank	time(s)	rank	time(s)	rank	time(s)
M_1	1e-2	15	1.19	9.3e-3	15	3.13	15		115	1.1
	1e-4	327	8.29	9.98e-5	328	3.71	313	123	990	9.7
M_2	1e-4	66	2.16	8.37e-5	66	2.73	65		101	1.1
	1e-5	82	2.68	8.86e-6	82	3.17	81	115	113	1.1
M_3	1e-2	33	1.56	4.1e-3	33	3.62	32		990	9.6
	1.5e-3	1588	18.7	1.499e-3	1587	15.8	1587	126	1990	37

We then test the algorithms with two actual matrices. One is from a scenic image in JPEG format [15], and the other is from an information retrieval application “AMiner” [16]. The colored image is represented by a 9504×4752 matrix. The second matrix is a 8130×100000 keyword-person matrix produced with the term frequency and inverse document frequency (TF-IDF) model [7]. This sparse matrix has about 0.2% nonzero elements. The results are shown in Table 4, with different settings of the power scheme and block size. They again validate that the proposed algorithms can automatically satisfy the accuracy threshold. And, with $P = 2$ the result of rank is substantially reduced, approaching the optimal value. For the same power scheme, setting larger block size b we can reduce the runtime of `randQB_EI`. In contrast, the runtime of `randQB_FP` increases with the block size, as we’ve set $\tilde{l} = 50b$. Notice that with the relative error $\delta = 0.1$, the image has little loss of quality, but is largely compressed ($\sim 7X$ size reduction). The comparison of original image and compressed image is shown in Figure 10. For the matrix from “AMiner”, the singular value presents very slow decay, but even with large approximation error the low-rank approximation could bring improved retrieval performance (c.f. [7], Sect. 11.3).

TABLE 4
The results of auto-rank problems based on two actual matrices

Matrix	δ	parameters	randQB_EI			randQB_FP		truncated SVD		RangeFinder [2]	
			rank	time(s)	error	rank	time(s)	rank	time(s)	rank	time(s)
image	0.1	$P=1, b=10$	468	8.1	0.0999	471	3.25	426	44.2	2913	79.0
		$P=1, b=20$	468	4.23	0.0999	472	4.44				
		$P=2, b=10$	441	9.98	0.0999	443	3.47				
		$P=2, b=20$	441	5.76	0.0999	443	7.26				
AMiner	0.5	$P=1, b=50$	2440	108	0.4999	2449	143	2115	1049	4990	192
		$P=2, b=50$	2229	134	0.4999	2242	205				

Because the second matrix is a large sparse matrix, one may think of using specific

technique for computing its singular values, i.e. the Krylov subspace based iterative method [5, 7]. In Matlab, command “svds” includes such technique. However, when we try it for the matrix we see that it costs 2281 seconds for computing the first 1000 singular values and singular vectors. It is much slower than executing “svd” to the matrix’s dense version. Besides, the SVD costs more than 20 GB memory, while the proposed randomized algorithms only costs 3 GB memory or so for this case.

To summarize the numerical results, we have the following remarks:

- The randQB_EI and randQB_FP algorithms with the power scheme both produce close to optimal low-rank QB factorization. They can bring 2X~3X reduction of runtime and memory usage compared with the blocked randQB algorithm [1], and have more benefits for handling large, and sparse matrices.
- For dense matrices, the randQB_FP is usually more efficient than the randQB_EI provided that the \tilde{l} is not largely overestimated. With the rank increasing, the result quality of randQB_FP is degraded. As the runtime of randQB_FP mainly depends on \tilde{l} , an approach for choosing a good \tilde{l} should be explored.
- The efficiency of the randQB_EI is more stable than the randQB_FP. For sparse matrices, the randQB_EI algorithm usually runs faster. And, the rank it produces is closer to the optimum.
- The randQB_FP algorithm may be preferable for handling extremely large matrix, because it takes fewer passes over the matrix.

8. Conclusions. The improvements to the blocked randQB scheme in [1] are presented. They produce two randomized low-rank factorization algorithms: randQB_EI and randQB_FP. The randQB_EI algorithm is about 2X more efficient than the blocked randQB in terms of runtime and memory usage, without loss of accuracy for usual low-rank approximation problems. The randQB_FP includes only $2P+1$ passes over matrix \mathbf{A} , where P is an integer usually less than 3, and has higher performance than the randQB_EI for handling a dense matrix.

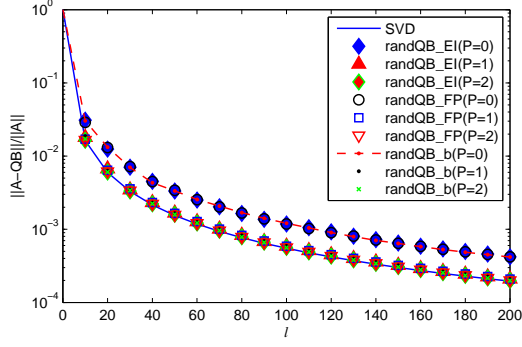
The proposed algorithms can be used for producing standard low-rank factorizations, like a partial QR factorization or a partial SVD. They are suitable for sparse and/or large matrices, and adaptive to the automatic rank determination problems in practice. In the future, the implementation of the proposed techniques on GPUs and their application to the problem with streaming data will be explored.

Acknowledgments. The authors thank Prof. P.-G. Martinsson for sharing the source code of the blocked randQB algorithm, which makes this work possible.

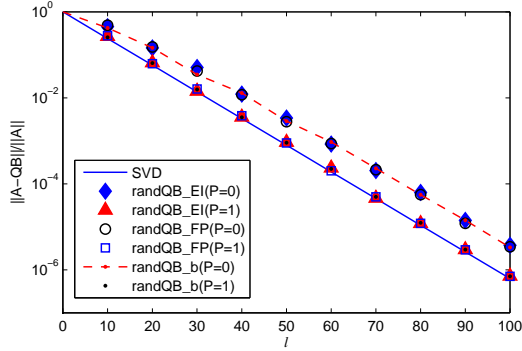
REFERENCES

- [1] P.-G. MARTINSSON AND S. VORONIN, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, SIAM J. Sci. Comput., accepted. arXiv:1503.07157v2 [math.NA], 2015.
- [2] N. HALKO, P.-G. MARTINSSON AND J. A. TROPP, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), no. 2, pp. 217–288.
- [3] S. VORONIN AND P.-G. MARTINSSON, *RSVDPACK: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and GPU architectures*, arXiv preprint, arXiv:1502.05366 [math.NA], 2015.
- [4] M. GU AND S. C. EISENSTAT, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), no. 4, pp. 848–869.
- [5] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
- [6] C. ECKART AND G. YOUNG, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

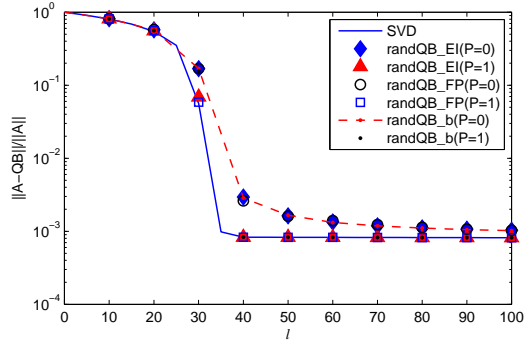
- [7] L. ELDEN, *Matrix Methods in Data Mining and Pattern Recognition*, SIAM Press, Philadelphia, MD, 2007.
- [8] S. ERIKSSON-BIQUE, M. SOLBRIG, M. STEFANELLI, S. WARKENTIN, R. ABBEY, AND I. C. F. IPSEN, *Importance sampling for a Monte Carlo matrix multiplication algorithm, with application to information retrieval*, SIAM J. Sci. Comput., 33 (2011), no. 4, pp. 1689–1706.
- [9] P. DRINEAS, R. KANNAN, AND M. W. MAHONEY, *Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix*, SIAM J. Sci. Comput., 36 (2006), pp. 158–183.
- [10] Q. YE, L. LUO, AND Z. ZHANG, *Frequent direction algorithms for approximate matrix multiplication with applications in CCA*, In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI’2016), 2016, 7 pages.
- [11] V. ROKHLIN, A. SZLAM, AND M. TYGERT, *A randomized algorithm for principal component analysis*, SIAM J. Matrix Anal. Appl., 31 (2009), no. 3, pp. 1100–1124.
- [12] S. VORONIN AND P.G. MARTINSSON, RandQR, http://amath.colorado.edu/faculty/martinss/main_codes.html, 2015.
- [13] Intel Parallel Studio XE Cluster Edition for Linux, <https://software.intel.com/en-us/intel-parallel-studio-xe>, 2016.
- [14] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM Press, Philadelphia, MD, 2003.
- [15] <http://flickr.com/photos/38459790@N03/4398318416>.
- [16] <http://www.aminer.org>.
- [17] H. JI AND Y. LI, *GPU accelerated randomized singular value decomposition and its application in image compression*, In Proceedings of Modeling, Simulation, and Visualization Capstone Conference, Suffolk, VA, 2014, 7 pages.
- [18] T. MARY, I. YAMAZAKI, J. KURZAK, P. LUSZCZEK, S. TOMOV, AND J. DONGARRA, *Performance of random sampling for computing low-rank approximations of a dense matrix on GPUs*, In Proc. SC’2015, Austin, TX, Nov. 2015, 11 pages.



(a) Errors of Matrix 1 whose singular values decay slowly



(b) Errors of Matrix 2 whose singular values decay rapidly



(c) Errors of Matrix 3 with S-shape decay of singular values

FIG. 8. Errors of different techniques for approximating the three test matrices ($n = 2000$, $b = 10$).

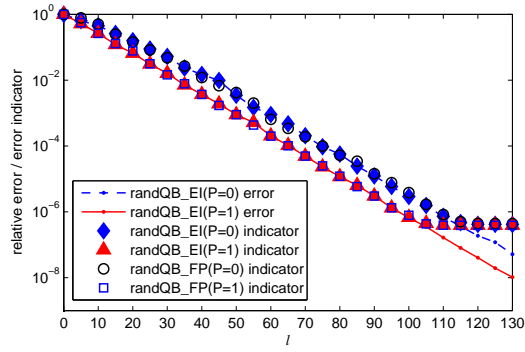


FIG. 9. The relative errors of the QB approximation and the values of error indicator in the proposed algorithms for Matrix 2.



(a) original image



(b) compressed image (1% error)

FIG. 10. The original image and the compressed image obtained with the proposed algorithm.